***Rapport TP noté n°1***

1. **Introduction**

Ce compte rendu a pour but de présenter le travail que j’ai effectué pour ce TP, qui avait pour but de créer les classes « Fraction » et « Liste chainée ».

Ce document me permet de détailler et expliquer les choix que j’ai fait vis-à-vis de l’implémentation de ces classes.

Cette dernière était dans un premier temps destiné à stocker des valeurs de type entier, puis je l’ai rendu générique afin qu’elle puisse stocker d’autres types de valeur, comme des objets de type « Fraction », des réels etc.

J’ai réalisé l’ensemble de ce code grâce au logiciel Visual Studio 2022.

De plus, ce TP est contenu dans une Solution nommée « TP\_note », contenant 3 projets :

« fraction », « liste\_chainee » et « liste\_chainee\_gen ».

Ces 3 projets correspondent respectivement à l’implémentation de : la classe « Fraction », la classe « Liste chainée » et la classe « Liste chainée » rendue générique.

Chacun de ces projets contient un « menu » présent dans le programme principal, permettant de tester les différentes méthodes implémentées.

1. **La classe « Fraction »**

Permet de créer des objets de type fraction, qui sont attribués d’un numérateur et d’un dénominateur.

**Indications sur les méthodes**

Méthode inverse : Cette méthode modifie directement l’objet qui a permis de l’appelé, donc ne renvoie pas un objet correspondant à l’inverse de l’objet ayant appelé cette méthode.

Méthode pgcd : Renvoie le pgcd de deux fractions en utilisant l’algorithme d’Euclide.

Méthode simplifier : Divise le numérateur et le dénominateur par le pgcd de ces derniers. De plus, si le dénominateur est négatif, je multiplie ces deux éléments par -1 pour que ce soit le numérateur qui soit négatif (plus cohérent).

**Gestion des exceptions**

Pour cette classe, il y’a eu 2 exceptions qui ont nécessité d’être traitées : le cas ou le dénominateur est nul et le cas ou on souhaite diviser une fraction par une fraction nulle.

1. **La classe « Liste chainée »**

Cette classe permet de stocker des nœuds en chaine.

Ces nœuds contiennent chacun deux éléments : un pointeur vers un nœud et un entier.

Chaque objet de cette classe est attribué d’une « tête de liste » : m\_debut, qui est un pointeur vers un nœud.

Étant donné que l’implémentation de cette classe fait intervenir l’allocation dynamique de mémoire, j’ai implémenté en plus du constructeur de base, un constructeur de recopie et un destructeur.

**Indications sur les méthodes**

Méthode insert : Pour cette méthode, je suis parti du principe que pour ajouter un élément à la position 1, il faut saisir i = 1, à l’inverse d’un tableau par exemple, ou le premier élément correspond à i = 0.

De plus, il n’est pas possible d’ajouter un élément à la position i si la liste contient i-1 éléments ou moins, car cela revient à faire un « push\_back ».

Méthode clear : Pour simplifier l’implémentation de cette méthode, j’ai décidé de me servir de la méthode « pop\_front », ce qui permet d’avoir un code plus épuré, et une complexité qui est toujours o(n).

**Problèmes rencontrés lors de l’implémentation de la classe**

Méthode pop\_back : nécessite de libérer de la mémoire le dernier nœud, et faire pointer le pointeur de l’avant dernier nœud vers « NULL ».

Pour ce faire, il faut manipuler le pointeur de l’avant-avant dernier nœud, pour pouvoir libérer la mémoire mais aussi modifier le contenu de l’avant dernier nœud, c’est pour ça que la condition d’arrêt de la boucle while est « temp->suivant->suivant == NULL ».

« temp » étant un pointeur vers un nœud permettant de traverser la liste chainée.

De plus, il faut gérer le cas ou la liste contient 1 élément car « temp » est initialisé à « m\_debut » et donc on ne pourra pas accéder à « temp ->suivant->suivant» car

« temp->suivant = NULL ».

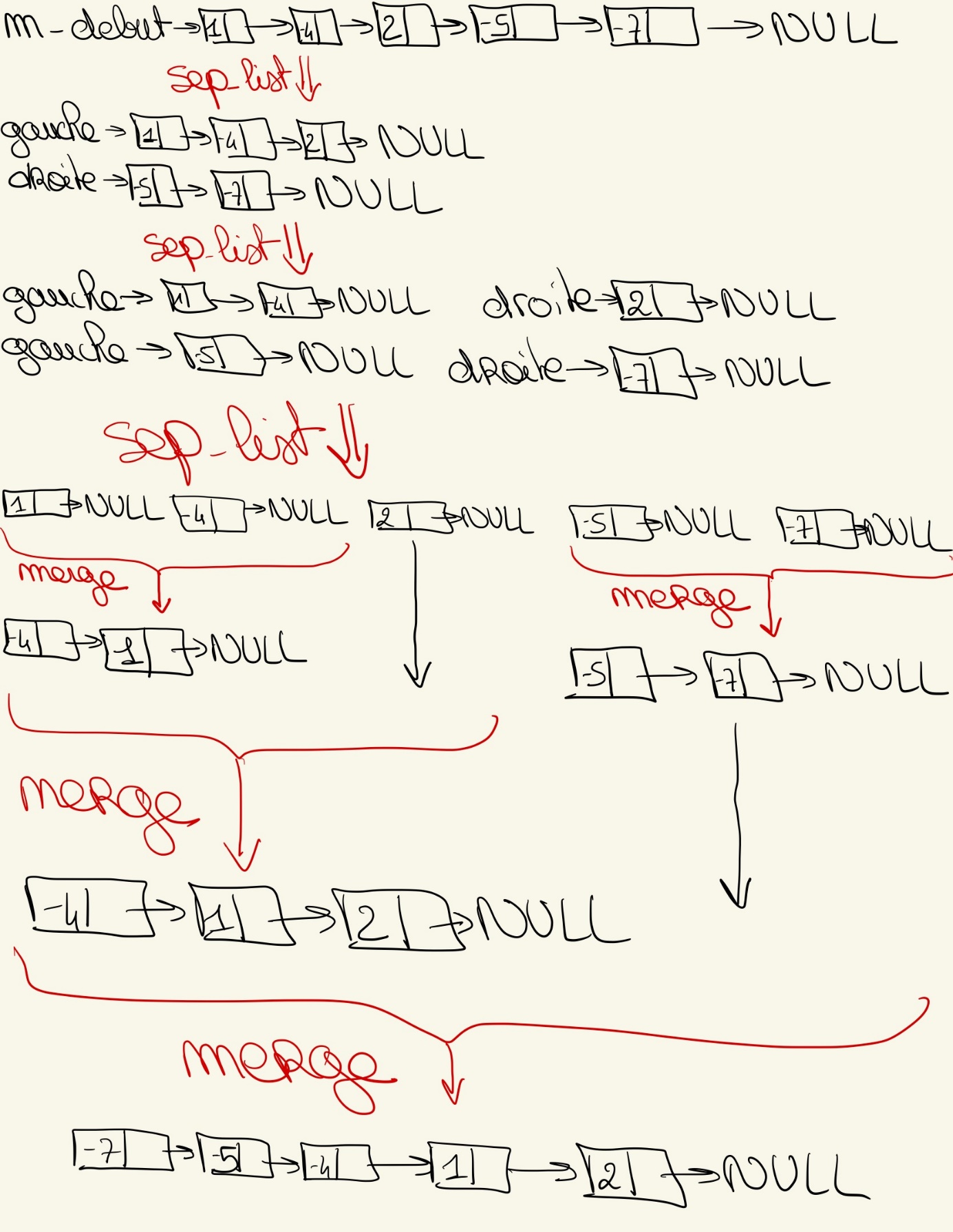
Une image contenant texte, écriture manuscrite, Police, calligraphie

Description générée automatiquement

**Tri par fusion**

J’ai décidé d’implémenter l’algorithme du tri par fusion pour la méthode « sort ». En effet, c’est l’algorithme qui présente la complexité la plus faible : o(nlogn).

Le principe est le suivant : on sépare récursivement la liste jusqu’à obtenir des listes d’au plus 1 élément chacune. Ensuite, on fusionne (merge) ces listes de 1 élément, et on obtient donc des listes de deux éléments (triées). On applique ce processus jusqu’à retomber sur la liste contenant tous les éléments, mais triée.



La première étape du tri est implémentée grâce à la méthode « sep\_list ».

Elle prend en argument la tête d’une liste contenant au moins deux éléments, et renvoie un pointeur vers la deuxième moitié de la liste.

Dans le cas où la liste contient un nombre d’éléments impair, la première moitié obtenue contient 1 élément de plus que la deuxième.

Pour bien diviser la liste en 2 moitiés, il faut donc faire pointer le pointeur du dernier nœud de la première moitié vers « NULL ».

Une image contenant texte, écriture manuscrite, Police, capture d’écran

Description générée automatiquement

Ensuite, dans la méthode récursive « merge\_sort », on attribue la première moitié obtenue à la liste chainée « gauche », et la deuxième à la liste « droite ».

On applique ensuite la méthode « merge\_sort » à ces deux listes : c’est l’application de la récursivité.

Quand on obtient deux listes de taille 1, on appelle la méthode « merge » en lui passant en paramètre les têtes des listes vers les deux moitiés « gauche » et « droite ».

Cette méthode renvoie un pointeur ces deux listes fusionnées et triées.

Ce pointeur est égal à « gauche » si la première valeur de la première moitié est inférieure ou égale à la première valeur de la deuxième moitié, et est égal à « droite » sinon.

Une image contenant texte, écriture manuscrite, Police, ligne

Description générée automatiquement

**Gestion des exceptions**

ExceptionListeVide : Problème lors de l’appel des méthodes « front », « back », « pop\_front » et « pop\_back » : il est impossible d’obtenir ou le supprimer le premier/dernier élément de la liste si cette dernière est vide.

ExceptionIndex : Problème lorsqu’on essaye d’accéder à un indice « out of range » d’une liste (i.e.. lorsqu’on utilise un indice « i » supérieur à la taille de la liste) pour les méthodes « insert » et « erase ».

1. **La classe « Liste chainée » générique**

L’implémentation de cette nouvelle classe a été assez rapide : il a suffit de combiner les deux précédents projets, en incluant dans le fichier d’entête initiant la classe «liste\_chainee » les implémentations des méthodes de la classe.